public interface BinaryTreeInterface<B> {
    int size();
    boolean isEmpty();
    boolean contains(B element);
    void insert(B element);
    int height();
    B remove(B element);
}
Pseudo Code

insertRec(node, key):
    if node == null:
        add key to tree
    if node.key > key:
        node.left =
        insertRec(node.left, key)
    else
        node.right =
        insertRec(node.right, key)
public void insert(E element) {
    size++;
    if (root == null) {
        root = new Node(element);
        return;
    }
    iInsert(root, element);
}

private void iInsert(Node treepart, E toBeAdded) {
    ...
}
public void insertAlt(E element) {
    if (root == null) {
        root = new Node(element);
        size = 1;
    } else
        iInsertAlt(root, element);
}
private void iInsertAlt(Node treepart, E toBeAdded) {
    int cmp = treepart.payload.compareTo(toBeAdded);
    if (cmp==0) return;   // the item is in the tree
    if (cmp<0) {
        if (treepart.left==null) {
            size++;
            treepart.left=new Node(toBeAdded);
        } else {
            iInsertAlt(treepart.left, toBeAdded);
        }
    } else {
        // cmp>0
        if (treepart.right==null) {
            size++;
            treepart.right=new Node(toBeAdded);
        } else {
            iInsertAlt(treepart.right, toBeAdded);
        }
    }
}
height()

```java
public int height()
{
    return iMaxDepth(root) - 1;
}

private int iMaxDepth(Node node)
{
    if (node == null) return 0;
    int rd = iMaxDepth(node.right) + 1;
    int ld = iMaxDepth(node.left) + 1;
    if (rd > ld)
        return rd;
    else
        return ld;
}
```
Binary Tree Traversals

• Traversal visits all nodes in a tree in some order
• Inorder: left subtree, current, right subtree
• Preorder: current, left subtree, right subtree
• Postorder: left subtree, right subtree, current
Traversals / Printing
public void printPostOrder() {
    iPrintPostOrder(root, 0);
    System.out.println();
}

private void iPrintPostOrder(Node treePart, int depth) {
    if (treePart==null) return;
    iPrintPostOrder(treePart.left, depth+1);
    iPrintPostOrder(treePart.right, depth+1);
    System.out.print("["+treePart.payload+","+depth+"]");
}
Remove

- boolean remove(E element);

  returns true if element existed and was removed and false otherwise

- Cases
  - element not in tree
  - element is a leaf
  - element has one child
  - element has two children
Leaf

- Just delete
One child

- Replace with child – skip over like in linked list
Two Children

• Replace with in-order predecessor or in-order successor

• in-order predecessor
  ▫ rightmost child in left subtree
  ▫ max-value child in left subtree

• in-order successor
  ▫ leftmost child in right subtree
  ▫ min-value child in right subtree
Replace with Predecessor
Replace with Successor
Mini-homework

Build tree with this list
104, 164, 139, 93, 132, 197, 167, 197, 113, 74, 17, 73, 124, 19, 181, 104, 1, 45, 88, 176

show the final tree

then remove and show tree after each removal for each of the following: numbers, (each removal follows from the previous). You need not redraw the entire tree after each deletion if you think the result is clear.

45
73
74
93
139
197
104

Send me a picture of your trees: gtowell206@cs.brynmawr.edu